|  |  |  |
| --- | --- | --- |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное автономное  образовательное учреждение высшего образования  «Пермский государственный национальный  исследовательский университет» | | |
|  | Институт компьютерных наук и технологий | |
| **ОТЧЁТ**  по индивидуальной работе №1  по дисциплине «Языки программирования»  “PGUCOIN” | | |
|  | | Работу выполнили  студенты группы ИТ-/1112-2024 1 курса  Мастыко М.В., Занадворных Н.М.  «25» Июня 2025 г. |
| Работу проверил  Кнутова Н.С.  «26» Июня 2025 г. |
| Пермь 2025 | | |

Оглавление

[Постановка задачи 3](#_Toc201765407)

[Основные цели и задачи проекта: 3](#_Toc201765408)

[Введение в технологию блокчейн 3](#_Toc201765409)

[Алгоритм решения 4](#_Toc201765410)

[3.1 Основные классы и структуры данных 5](#_Toc201765411)

[3.2. Ключевые процессы и жизненный цикл системы 7](#_Toc201765412)

[4. Описание функционала программы 9](#_Toc201765413)

[Тестирование 10](#_Toc201765414)

[Код программы 10](#_Toc201765415)

# Постановка задачи

Требовалось разработать программное приложение на языке C++, моделирующее работу упрощенной криптовалютной системы на основе технологии блокчейн. Проект получил название PGUCOIN.

## Основные цели и задачи проекта:

1. Реализация базовых компонентов блокчейна:

* Создание структуры Block для хранения транзакций, временной метки, хеша предыдущего блока и служебной информации (nonce).
* Формирование непрерывной цепи блоков (BlockChain), где каждый последующий блок криптографически связан с предыдущим.

2. Система транзакций:

* Создание класса Transaction для описания операций перевода средств.
* Реализация пула (Pool) для временного хранения неподтвержденных транзакций.

3. Моделирование майнинга:

* Реализация механизма Proof-of-Work (PoW), в рамках которого "майнеры" решают вычислительную задачу для получения права на добавление нового блока в цепь.
* Внедрение понятия сложности майнинга.

4. Управление пользователями:

* Создание класса User для представления участников сети.
* Реализация возможности генерации пользователей и выполнения ими транзакций (депозит, вывод, перевод).

5. Интерфейс и персистентность:

* Разработка консольного меню для взаимодействия с системой.
* Реализация механизма сохранения состояния блокчейна в файл и его последующей загрузки для обеспечения непрерывности данных между сессиями.

# Введение в технологию блокчейн

Блокчейн (англ. blockchain, "цепь блоков") — это распределенная база данных, которая представляет собой непрерывную последовательную цепочку блоков, содержащих информацию. Эта технология лежит в основе большинства криптовалют, включая Bitcoin.

Ключевые принципы и компоненты блокчейна, реализованные в данном проекте:

1. Блок: Основная структурная единица блокчейна. В проекте PGUCOIN каждый Block содержит:

* Список транзакций: Записи об операциях, включенных в блок.
* Хеш предыдущего блока: Криптографическая ссылка, которая связывает текущий блок с предыдущим, образуя цепь. Это обеспечивает неизменность истории: изменение информации в одном из прошлых блоков привело бы к изменению его хеша, что разорвало бы всю последующую цепь.
* Временная метка (Timestamp): Время создания блока.
* Nonce (Number used once): "Одноразовое число", которое майнеры подбирают в процессе PoW для нахождения валидного хеша блока.
* Корень Меркла (Merkle Root): Единый хеш, который агрегирует хеши всех транзакций в блоке. Это позволяет эффективно и быстро проверять, была ли конкретная транзакция включена в блок, не перебирая их все.

2. Proof-of-Work (PoW): Алгоритм консенсуса, используемый для защиты сети от атак и двойных трат. Суть его заключается в необходимости выполнения сложной, ресурсоемкой вычислительной задачи. В нашем проекте это поиск такого значения Nonce, при котором хеш блока (рассчитанный по алгоритму SHA-256) будет начинаться с определенного количества нулей (сложность майнинга). Первый, кто находит решение, получает право добавить новый блок в цепь.

3. Хеширование: В проекте используется криптографическая хеш-функция SHA-256. Она преобразует любой объем входных данных в строку фиксированной длины (256 бит). Основные свойства:

* Детерминированность: Один и тот же ввод всегда дает один и тот же хеш.
* Необратимость: восстановить исходные данные из хеша практически невозможно.
* Лавинный эффект: Минимальное изменение входных данных кардинально меняет выходной хеш.

# Алгоритм решения

Для реализации модели блокчейн-системы PGUCOIN была выбрана объектно-ориентированная парадигма. Архитектура проекта построена на разделении логических сущностей (пользователь, транзакция, блок) на отдельные классы. Это обеспечивает высокую модульность, упрощает понимание кода и его дальнейшее расширение. Взаимодействие между объектами моделирует ключевые процессы, происходящие в реальных блокчейн-сетях.

## 3.1 Основные классы и структуры данных

Проект состоит из нескольких ключевых классов, каждый из которых отвечает за свою часть общей логики.

* User (Пользователь)
* Назначение: представляет участника или узел в сети PGUCOIN. Каждый пользователь имеет уникальное имя и идентификатор.
* Ключевые атрибуты:

1. UserID: int — Уникальный числовой идентификатор. ID=0 зарезервирован для системного пользователя Burse.
2. Name: std::string — Имя пользователя, генерируемое случайным образом для удобства идентификации.
3. HisBlockChain: BlockChain\* — Указатель на экземпляр блокчейна. Эта связь необходима, чтобы пользователь мог отправлять транзакции в пул своей сети.

* Ключевые методы:

1. MakeTransaction(User\* Person, double Amount, Action Oper): Это основной метод, с помощью которого пользователь инициирует транзакцию. В зависимости от параметра Oper (DEPOSIT, WITHDRAW, TRANSFER), он создает объект Transaction с правильными отправителем и получателем (например, при депозите отправителем является Burse) и добавляет его в пул неподтвержденных транзакций (TransactionsPool) через указатель HisBlockChain.

* Transaction (Транзакция)
* Назначение: Атомарная единица операции в системе. Описывает перевод средств от одного участника к другому.
* Ключевые атрибуты:

1. Sender: User\* — Указатель на отправителя средств.
2. Receiver: User\* — Указатель на получателя средств.
3. Amount: double — Сумма перевода.
4. TimeStamp: time\_t — Временная метка, фиксирующая время создания транзакции.

* Ключевые методы:

1. ToString() const: Критически важный метод, который сериализует данные транзакции (ID отправителя, ID получателя, сумма, временная метка) в уникальную строку. Эта строка используется для вычисления хеша транзакции при построении дерева Меркла.

* Pool (Пул транзакций)
* Назначение: Временное хранилище для транзакций, которые были созданы пользователями, но еще не были включены в блок. В реальных системах это называется "mempool".
* Структура данных: Реализован на основе std::queue(очереди), что обеспечивает обработку транзакций в порядке их поступления (FIFO — First-In, First-Out).
* Ключевые методы:

1. AddTrans(Transaction\* AddedTrans): Добавляет новую транзакцию в конец очереди.
2. TopTrans(): Возвращает самую старую транзакцию из пула без ее удаления.
3. DeleteTrans(): Удаляет самую старую транзакцию из пула.

* Block (Блок)
* Назначение: Основная структура данных, из которой состоит блокчейн. Блок является контейнером для группы транзакций и служебной информации.
* Ключевые атрибуты:

1. Number: int — Порядковый номер блока в цепи (высота блока).
2. PrevBlock: Block\* — Указатель на предыдущий блок. Именно это поле криптографически связывает блоки в единую цепь. Для генезис-блока это поле равно nullptr.
3. TransactionList: std::vector<Transaction\*> — Список транзакций, включенных в данный блок.
4. HashMerkle: std::string — Корень дерева Меркла. Это единый хеш, агрегирующий хеши всех транзакций в блоке. Он позволяет эффективно проверять целостность списка транзакций и наличие в нем конкретной операции.
5. TimeStamp: time\_t — Время "закрытия" блока (нахождения решения PoW).
6. Nonce: int — "Number used once". Случайное число, которое подбирается в процессе майнинга для нахождения валидного хеша блока.
7. Hash: std::string — Хеш текущего блока, вычисленный на основе всех его данных, включая Nonce.

* Ключевые методы:

1. CalculateMerkleRoot(const std::vector<Transaction\*>& Transactions): Статический метод, который строит дерево Меркла. Он берет хеши всех транзакций, попарно хеширует их, пока не останется один-единственный хеш — корень.
2. CalculateHash(): Вычисляет хеш блока по алгоритму SHA-256 на основе строки, полученной из метода ToString().
3. ToString() const: Сериализует ключевые поля блока (номер, хеш пред. блока, корень Меркла, timestamp, nonce) в строку для последующего хеширования.

* BlockChain (Цепь блоков)
* Назначение: Центральный управляющий класс, который инкапсулирует всю логику работы с цепью блоков.
* Ключевые атрибуты:

1. GenesisBlock: Block\* — Указатель на самый первый (нулевой) блок.
2. LastBlock: Block\* — Указатель на последний добавленный блок, что позволяет быстро добавлять новые блоки в конец цепи.
3. TransactionsPool: Pool\* — Указатель на пул неподтвержденных транзакций.
4. CurrentHardMining: int — Целочисленное значение, определяющее сложность майнинга (количество нулей, с которых должен начинаться хеш блока).
5. AllUsers: std::vector<User\*>\* — Указатель на вектор всех зарегистрированных в системе пользователей.

* Ключевые методы:

1. MineBlock(): Реализует процесс майнинга.
2. GetUserBalance(const User\* CurUser) const: Рассчитывает и возвращает баланс указанного пользователя.
3. IsTransactionValid(Transaction\* trans) const: Проверяет, может ли транзакция быть выполнена (например, достаточно ли средств у отправителя).
4. SaveToFile() / LoadFromFile(): Отвечают за персистентность — сохранение всего состояния блокчейна (пользователей, блоков, транзакций) в бинарный файл и его загрузку при старте программы.

## 3.2. Ключевые процессы и жизненный цикл системы

1. Инициализация и загрузка:
2. При запуске main() создается экземпляр BlockChain.
3. Вызывается метод LoadFromFile(). Если файл pgucoin.dat существует и корректен, все состояние системы (пользователи, блоки и их связи) восстанавливается из него.
4. Если файл не найден, система инициализируется с нуля: генерируются новые пользователи, создается генезис-блок (CreateGenesisBlock), и начальное состояние сохраняется в файл.

2. Создание и обработка транзакции:

1. Пользователь через меню выбирает тип операции (депозит, вывод, перевод).
2. Программа запрашивает необходимые данные (пользователей, сумму).
3. Вызывается метод User::MakeTransaction().
4. Внутри этого метода создается объект Transaction и помещается в TransactionsPool, ожидая подтверждения.

3. Майнинг нового блока (Цикл Proof-of-Work): Это центральный процесс, который обеспечивает безопасность и пополнение цепи.

1. Пользователь запускает майнинг из меню. Вызывается метод BlockChain::MineBlock().
2. Метод извлекает транзакции из TransactionsPool.
3. Каждая транзакция проверяется на валидность с помощью IsTransactionValid(). В частности, для операций списания (WITHDRAW, TRANSFER) вызывается GetUserBalance(), чтобы убедиться, что у отправителя достаточно средств. Невалидные транзакции отбрасываются.
4. Из валидных транзакций формируется список, для которого вычисляется корень Меркла HashMerkle.
5. Создается объект-кандидат нового блока NewBlock, куда помещаются все необходимые данные.
6. Запускается цикл Proof-of-Work:

* while (!CheckSolution(Solution)) — цикл продолжается, пока не будет найдено решение.
* Внутри цикла инкрементируется NewBlock->RiseNonce().
* Вычисляется новый хеш блока sha256(NewBlock->ToString()).
* CheckSolution проверяет, начинается ли полученный хеш с нужного количества нулей (CurrentHardMining).

1. Как только решение найдено, цикл завершается. Блок считается "добытым".
2. Новый блок связывается с цепью: LastBlock->SetNextBlock(NewBlock). Указатель LastBlock смещается на NewBlock.
3. Счетчик блоков CountBlocks увеличивается.
4. Обновленное состояние блокчейна немедленно сохраняется в файл через SaveToFile().

4. Расчет баланса и отображение цепи:

1. При запросе баланса пользователя (GetUserBalance) система не хранит готовое значение. Вместо этого она проходит по всей цепи от GenesisBlock до LastBlock, анализируя каждую транзакцию в каждом блоке. Если пользователь является получателем, сумма добавляется к его балансу, если отправителем — вычитается. Это гарантирует, что баланс всегда соответствует неизменной истории транзакций.
2. При отображении блокчейна (PrintChain) программа аналогично итерирует по всей цепи с помощью указателей NextBlock и выводит подробную информацию о каждом блоке.

# 4. Описание функционала программы

Программа предоставляет пользователю текстовый интерфейс (CLI) для управления симуляцией блокчейна.

Главное меню:

* 0. | Mine a Block |: запускает процесс майнинга нового блока. В него войдут все транзакции, находящиеся в пуле на данный момент. После успешного майнинга состояние автоматически сохраняется.
* 1. Make a Deposit: позволяет пополнить баланс выбранного пользователя. Транзакция создается от системного пользователя Burse.
* 2. Make a Withdrawal: позволяет списать средства с баланса пользователя. Транзакция уходит системному пользователю Burse.
* 3. Make a Transfer: осуществляет перевод средств между двумя выбранными пользователями.
* 4. View User Balances: отображает текущие балансы всех пользователей, включая системного Burse. Баланс рассчитывается путем итерации по всей цепи блоков.
* 5. View Blockchain: выводит в консоль полную информацию о каждом блоке в цепи: его номер, хеш, хеш предыдущего блока, корень меркла, временную метку и список транзакций.
* 6. Exit: корректно завершает работу программы. Все данные, сохраненные после последнего майнинга, остаются в файле pgucoin.dat.
* 7. Clear Blockchain and Exit: удаляет файл pgucoin.dat, фактически сбрасывая блокчейн к начальному состоянию при следующем запуске, и завершает программу.

Управление пользователями:

При первом запуске программа генерирует заданное количество пользователей со случайными именами из файлов data/FirstNames.txt и data/SecondNames.txt. При последующих запусках пользователи и их история загружаются из файла сохранения.

# Тестирование

<https://youtu.be/P51alrq4pjY>

# Код программы

<https://github.com/MineMoon/PGUCOIN.git>